A More Reliable Greedy Heuristic for Maximum Matchings in Sparse Random Graphs*

Martin Dietzfelbinger[1], Hendrik Peilke[2,**], and Michael Rink[1]

[1] Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau
{martin.dietzfelbinger,michael.rink}@tu-ilmenau.de
[2] IBYKUS AG für Informationstechnologie, Erfurt, Germany
Hendrik.Peilke@Ibykus.de

Abstract. We propose a new greedy algorithm for the maximum cardinality matching problem. We give experimental evidence that this algorithm is likely to find a maximum matching in random graphs with constant expected degree $c > 0$, independent of the value of c. This is contrary to the behavior of commonly used greedy matching heuristics which are known to have some range of c where they probably fail to compute a maximum matching.

1 Introduction

Maximum Cardinality Matchings. Consider an undirected graph $G = (V, E)$ with node set V, $|V| = n$, and edge set $E \subseteq \binom{V}{2}$, $|E| = m$. A matching M in G is a subset of E with the property that the edges in M are pairwise disjoint. The problem of finding a matching with the largest possible cardinality, a so called *maximum matching*, has been a subject of study for decades. The first polynomial time algorithm for this problem was given in 1965 by Edmonds [7]. A straightforward implementation of this algorithm has running time $O(n^2 \cdot m)$. Many other polynomial time algorithms followed, eventually reducing the running time to $O(n^{1/2} \cdot m)$, as, e.g., the algorithm of Micali and Vazirani [12,16]. For dense graphs, i.e., graphs with $m = \Theta(n^2)$ edges, this was the best known until 2004 when Mucha and Sankowski [13] gave an algorithm that has (expected) running time dominated by the time for multiplying two $n \times n$ matrices, which is $O(n^\omega)$, with $\omega < 2.376$ [5].

Heuristics. Usually matching algorithms, notably augmenting path algorithms, are allowed to be initialized with a non-empty matching which is then iteratively improved to a maximum matching. Hence a large enough initial matching determined with some fast heuristic approach can decrease the running time of an exact algorithm significantly. Beyond the use of heuristics in the preprocessing phase of exact algorithms, there is an interest in graph classes where heuristics, especially fast greedy algorithms, are likely to obtain maximum matchings. On

* Research supported by DFG grant DI 412/10-2.
** M.Sc. student at the Technische Universität Ilmenau while this work was done.

such classes heuristics can replace the (overall) exact algorithms if the heuristics are faster or at least equally fast but easier to implement.

Sparse Random Graphs. A well studied graph class in this context is the class of random graphs with constant expected degree c. Let $G(n; c)$ be a random (general) graph with n nodes where each of the $\binom{n}{2}$ possible edges is present with probability $p = c/(n-1)$, and let $B(n/2, n/2; c)$ be a random bipartite graph with n nodes where each of the $n^2/4$ possible edges is present with probability $p = c \cdot 2/n$. Bast et al. [2] showed that if $c > c_0$ for $c_0 = 32.67$ in the case of general graphs, and $c_0 = 8.83$ in the case of bipartite graphs, then with high probability every non-maximum matching in $G(n; c)$ and $B(n/2, n/2; c)$ has an augmenting path of length $O(\log n)$. (Note that this trivially holds for $c \in (0, 1)$ and indeed it is conjectured that $c_0 = 0$ in both cases.) Hence matching algorithms using shortest augmenting paths like the algorithm of Micali and Vazirani for general graphs and the algorithm of Hopcroft and Karp [9] for bipartite graphs have (expected) running time $O(n \cdot \log n)$ on sparse random graphs. Chebolu et al. [4] gave an algorithm that improved the (expected) running time to $O(n)$ using a simple heuristic in the first phase of their algorithm, usually called Karp-Sipser algorithm. Karp and Sipser [10] proved that this greedy algorithm produces a matching which is within $o(n)$ of the maximum for every constant $c > 0$. This result was improved by Aronson et al. [1] who showed that actually for $c < e$ the Karp-Sipser algorithm finds a maximum matching with high probability and for $c > e$ the size of the matching is within $n^{1/5+o(1)}$ of the maximum. Interestingly, for practical purposes Karp and Sipser suggested a different greedy algorithm, Algorithm 1 of [10], that turns out to give better results in their experiments but seems to be much more complicated to analyze because it utilizes contraction of nodes.

"Critical Region". In an experimental study Magun [11] compared the performance of several greedy matching algorithms in the style of the algorithms given in [10] on sparse random graphs. It turned out that there are good greedy algorithms that are likely to give maximum matchings for a wide range of c, but even the best algorithm in this study fails in the range of about $2.6 \leq c \leq 3.8$ (where the lower bound is likely to converge to $e \approx 2.718$ for n large enough). Hence there is some region for c that seems critical for known greedy matching heuristics.

1.1 Our Results

We describe a new greedy matching algorithm and give experimental evidence that this algorithm is likely to compute a maximum matching in sparse random graphs for all ranges of c and large enough n; in particular, it seems to overcome the critical region mentioned in [11]. The algorithm is motivated by the "selfless algorithm" of Sanders [14], for orienting undirected graphs such that the maximum in-degree is below a given constant.

Drawback. In comparison to the common greedy heuristics discussed above the running time of our algorithm is larger and more affected by the expected degree c. Hence, we propose using a combined algorithm using our approach solely for the critical region.

1.2 Overview of the Paper

In the next section we consider several common greedy matching heuristics and give some motivation for our new approach. Following that, in the main part of the paper we describe the experiments and discuss the results.

2 Greedy Matching Heuristics

In this section we give a brief description of the greedy matching algorithms considered here. The structure of this section is similar to Section 3 of [11].

Basic Structure. The algorithms work recursively. Let $G_0 = G$ be the input graph. Consider some arbitrary recursion level $l \geq 0$. Let G_l be the current graph, and let d be the minimum degree of G_l. There are two cases:

$d \leq 2$. Apply an "optimal reduction step" on G_l, i.e., depending on d, remove nodes and edges from G_l to yield G_{l+1}.

$d \geq 3$. Apply a "heuristic reduction step" on G_l, i.e., choose an edge $e = \{u, v\}$ from G_l with the highest priority according to some heuristic order of priority, and remove u and v and all incident edges from G_l to yield G_{l+1}.

Run the algorithm recursively on G_{l+1}, which will return a matching M_{l+1} for G_{l+1}. Finally, add an edge to M_{l+1} to obtain a matching M_l for G_l. An optimal step will never decrease the size of a maximum matching, while a heuristic step might do that.

Optimal Steps. The two optimal steps that we consider are commonly known as "degree 1 reduction" and "degree 2 reduction". They are based on the following facts proved by Karp and Sipser in [10].

Fact 1. Let $G = (V, E)$ be a graph. If there exists a node $u \in V$ with degree $\deg(u) = 1$, adjacent to a node $v \in V$, then there exists a maximum matching M in G with $\{u, v\} \in M$.

Fact 2. Let $G = (V, E)$ be a graph. If there exists a node $u \in V$ with degree $\deg(u) = 2$, adjacent to nodes $v_1, v_2 \in V$, then there exists a maximum matching M in G with either $\{u, v_1\} \in M$ or $\{u, v_2\} \in M$.

For any subset V' of the nodes of G let $G \setminus V'$ be the subgraph of G that is induced by all nodes of $V \setminus V$' and let $G \circ V'$ be the graph that results from G by contracting all nodes of V' into a single node and removing all multiple edges and self-loops. Using these definitions we can state the optimal degree reduction steps as follows.

degree 1 reduction: Randomly choose a node u from G_l with degree $\deg(u) = 1$, incident to an edge e. Shrink the graph G_l via $G_{l+1} \leftarrow G_l \setminus e$. Increase the matching M_{l+1} given by the recursive call, via $M_l \leftarrow M_{l+1} \cup \{e\}$.

degree 2 reduction: Randomly choose a node u from G_l with degree $\deg(u) = 2$, adjacent to nodes v_1, v_2. Contract the three nodes into a single node v via $G_{l+1} \leftarrow G_l \circ \{u, v_1, v_2\}$ and store how v was constructed. If an edge $e = \{v, w\}$ is part of the matching M_{l+1} given by the recursive call, then, to obtain the matching M_l, either replace e with $\{v_1, w\}$ in M_{l+1} and add $\{u, v_2\}$ to M_{l+1}, or replace e with $\{v_2, w\}$ in M_{l+1} and add $\{u, v_1\}$ to M_{l+1}.

In the following we will use "OPT(1)" and "optimal degree 1 reduction", as well as "OPT(1,2)" and "optimal degree 1 and optimal degree 2 reduction" synonymously.

Heuristic Steps. The procedure of the heuristic step is similar to the degree 1 reduction step. First choose an edge e, then shrink the graph via $G_{l+1} \leftarrow G_l \setminus e$, and finally increase the matching via $M_l \leftarrow M_{l+1} \cup \{e\}$. The choice of the edge is based on a priority order of the edges, where the priorities are calculated using properties in the neighborhood of the nodes. We consider the following heuristics.

random edge: Randomly choose an edge $e \in E$.

double minimum degree: Randomly choose a node $u \in V$ among the nodes with smallest degree. Randomly choose an edge $e = \{u, v\} \in E$ where v is among the neighbors of u that have smallest degree.

minimum expected potential, minimum degree: Randomly choose a node $u \in V$ among the nodes with smallest potential $\pi(u)$, where

$$\pi(u) = \sum\nolimits_{\{u,v\} \in E} \frac{1}{\deg(v)} \ .$$

Then randomly choose an edge $e = \{u, v\} \in E$ where v is among the neighbors of u that have smallest degree.

Simply choosing an edge at random can be seen as all edges having the same priority, which disregards the structure of the graph. The idea of choosing a node of low degree is that the lower the degree the fewer the possibilities of the node u to be covered by a matching. This is taken one step further in the third heuristic by calculating the values $\pi(u)$. If each neighbor v of a node u randomly declares one of its incident edges to be the only edge that is allowed to cover v in a matching then the value $\pi(u)$ is the expected number of potential matching edges that could cover u. As before, the lower the number of possibilities the more urgent it is to include the node in a matching edge.

In the following we will use interchangeably: "HEU(rand)" and "random edge heuristic", "HEU(deg,deg)" and "double minimum degree heuristic", as well as, "HEU(pot,deg)" and "minimum expected potential, minimum degree heuristic".

Algorithms. We list six matching algorithms whose performance is experimentally examined in our experiments, where the last two algorithms are new. The

names of the algorithms are generic, describing their structure as combination of the utilized optimal and heuristic steps. If an algorithm uses OPT(1,2) then the degree 1 reduction step is always preferred to the degree 2 reduction step.

OPT(1):HEU(rand) This algorithm is commonly known as Karp-Sipser algorithm as it was first analyzed by Karp and Sipser in [10, Algorithm 2]. If the expected degree c of a sparse random graph is below e then the algorithm finds a maximum matching (with high probability) and if c is larger than e then the matching is within $n^{1/5+o(1)}$ of the maximum cardinality (with high probability), see [1].

OPT(1,2):HEU(rand) This is a variant of the Karp-Sipser algorithm using in addition the degree 2 reduction step, which was also proposed in [10]. It is included to investigate the effect of the degree 2 reduction.

OPT(1):HEU(deg,deg) This algorithm is recommended in the experimental study [11] as the most practical algorithm, see [11, Conclusion]. Note that the optimal degree 1 reduction needs not to be implemented separately since it is performed implicitly by the heuristic step.

OPT(1,2):HEU(deg,deg) This is one of the two algorithms proposed in [11] that offer the highest quality of solution. The other one (called BlockRed) is more complicated, using an additional optimal reduction, but has very similar performance. It was demonstrated experimentally that both algorithms are likely to compute a maximum matching in sparse random graphs when $c < 2.6$ or $c > 3.8$, but fail to do so for other values of c. Moreover, in the "critical region" $2.6 \leq c \leq 3.8$ the number of edges that are missing from a matching with maximum cardinality is increasing with increasing n.

OPT(1):HEU(pot,deg) This is the first new algorithm. It is a straightforward adaption of the selfless algorithm proposed by Sanders in [14] for determining an orientation of the edges of an undirected graph. The selfless algorithm has been proven to be optimal in the sense that with high probability it obtains an orientation of the edges of an undirected sparse random graph that gives minimum in-degree, if the density is such that such an orientation exists, see [3].

OPT(1,2):HEU(pot,deg) This is the second new algorithm and the outcome of our search for an algorithm that has probably no critical region. As shown in the following experiments the additional use of the degree 2 reduction is essential.

Note that the recursive structure of the algorithms can easily be transformed into an iterative structure, if there is no degree 2 reduction or one only needs to compute the size of a maximum matching, since in both cases there is no need to resolve contraction of nodes.

Algorithm OPT(1,2):HEU(pot,deg) is the heuristic that we propose for computing maximum cardinality matchings in sparse random graphs, therefore its pseudocode (Algorithm 1) is given below for completeness.

Algorithm 1: `OPT(1,2):HEU(pot,deg)`$[G$`: graph]`

Input: *simple graph* $G = (V, E)$ *with node set* V *and edge set* E
Output: *matching* M

$M \leftarrow \emptyset$;
if $E \neq \emptyset$ **then**
  $d \leftarrow$minimum degree of all nodes in V;
  if $d = 1$ **then**
    $u \leftarrow$ random node from V with $\deg(u) = 1$;
    $v \leftarrow$ neighbor of u;
    $M \leftarrow$ `OPT(1,2):HEU(pot,deg)`$[G \setminus \{u, v\}]$;
    $M \leftarrow M \cup \{\{u, v\}\}$;
  else if $d = 2$ **then**
    $u \leftarrow$ random node from V with $\deg(u) = 2$;
    $\{v_1, v_2\} \leftarrow$ set of 2 neighbors of u;
    $v \leftarrow \{u, v_1, v_2\}$;
    $M \leftarrow$ `OPT(1,2):HEU(pot,deg)`$[G \circ \{u, v_1, v_2\}]$;
    if v is not matched in M **then** $M \leftarrow M \cup \{\{u, v_1\}\}$;
    else
      $w \leftarrow$ matching neighbor of v;
      $M \leftarrow M \setminus \{\{v, w\}\}$;
      if $\{v_1, w\} \in E$ **then** $M \leftarrow M \cup \{\{v_1, w\}, \{u, v_2\}\}$;
      else $M \leftarrow M \cup \{\{v_2, w\}, \{u, v_1\}\}$;
  else
    $\pi \leftarrow$minimum potential of all nodes in V;
    $u \leftarrow$ random node from V with $\pi(u) = \pi$;
    $N \leftarrow$ set of neighbors of u;
    $v \leftarrow$ random node from N with minimum degree;
    $M \leftarrow$ `OPT(1,2):HEU(pot,deg)`$[G \setminus \{u, v\}]$;
    $M \leftarrow M \cup \{\{u, v\}\}$;
return M;

3 Experiments

We examine the performance of the six greedy matching algorithms, given in the last section, on random general graphs $G(n;c)$ and random bipartite graphs $B(n/2,n/2;c)$ with n nodes and constant expected average degree c. We cover parameter ranges $n \in \{10^4, 10^5, 10^6\}$ and $c \in [1,10]$, where parameter c is iteratively increased via $c = 1 + i \cdot 0.1$, for $i = 0, 1, \ldots, 90$.

Construction of Random Graphs. Let $N = \binom{n}{2}, p = c/(n-1)$ for random general graphs $G(n;c)$, and let $N = n^2/4, p = c \cdot 2/n$ for random bipartite graphs $B(n/2,n/2;c)$. For fixed parameters (n,c) the construction of a random graph $G = (V,E)$ is done as follows. We start with the node set $V = \{1,2,\ldots,n\}$ and an empty edge set E. If $n = 10^4$ then each of the N possible edges is generated and added to E with probability p independently of all other edges. If $n \in \{10^5, 10^6\}$ then, in order to keep the construction time manageable, we first determine the number of edges X, which is expected to be linear in n, and then randomly choose X edges from the set of N possible edges. The number of edges follows a binomial distribution $X \sim \mathrm{Bin}(N,p)$. To determine a realization x of X, we determine a realization y of a standard normal random variable $Y \sim \mathrm{Nor}(0,1)$ using the polar method [15, Section 2.3.1]. The value $\tilde{x} = \mathrm{round}(y \cdot \sqrt{N \cdot p(1-p)} + N \cdot p)$ is used as an approximation of x. As long as $\tilde{x}$ is not feasible the calculation is repeated with new realizations of N.

Measurements. For each pair of parameters (n,c) we constructed 100 random graphs (bipartite and general) and measured the following quantities for each of the six heuristics:

- the failure rate λ. This is the fraction of graphs where the matching obtained by the heuristic is not a maximum matching.
- the average number of "lost edges" ρ, which we define as the average number of edges missing from a maximum matching, conditioned on the event that a failure occurs. If no failure occurs we let $\rho = 0$.

To get insight in how the parameter c might influence the running time of our new algorithm we did additional experiments using random graphs with $n = 10^6$ nodes. For each c we constructed 10 random graphs (bipartite and general) and measured the following quantities for OPT(1,2):HEU(pot,deg):

- the average running time $\bar{t}$ needed to obtain a matching, as well as the corresponding sample variance.
- the average fraction of: degree 1 reduction steps $\overline{\#o1}$, degree 2 reduction steps $\overline{\#o2}$, and heuristic steps $\overline{\#h}$.

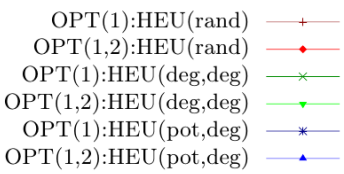
System. The source code for the graph generators as well as for the algorithms is written in C++ and compiled with g++ version 4.5.1. The experiments regarding the running time ran on an Intel Xeon CPU E5450 (using one core) under openSUSE with kernel 2.6.37.6-0.9-desktop.

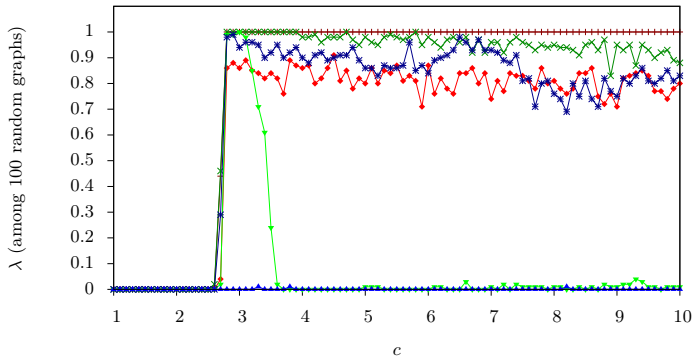
Random Source. For the necessary random choices for the algorithms as well as for the construction of the random graphs we used the pseudo random number generator MT19937 "Mersenne Twister" of the GNU Scientific Library [8].

3.1 Results

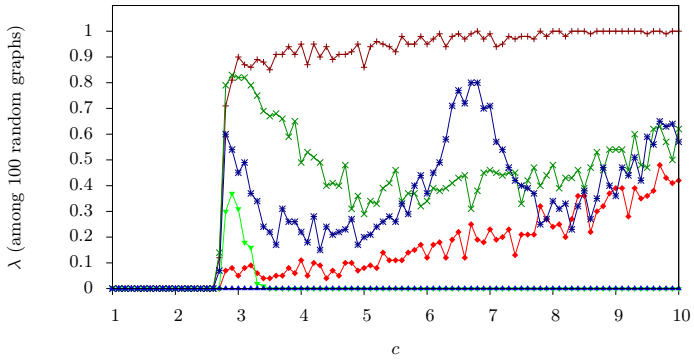
Here we consider results from the matching heuristics given in Section 2.

Failure Rates. Figure 1 gives the failure rates on general and bipartite random graphs with $n = 10^6$ nodes and expected degree c ranging from 1 to 10. The legend for both plots is given to the right. Figures depicting the failure rates for graphs with 10^4 and 10^5 nodes are given in Appendix A. The results are qualitatively similar to the results for $n = 10^6$.





(a) general graphs



(b) bipartite graphs

Fig. 1. Failure rates on graphs with $n = 10^6$ nodes.

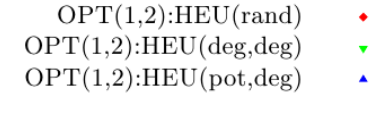
For $1 \leq c \leq 2.5$ no failure occurred in any of the algorithms. Our new algorithm OPT(1,2):HEU(pot,deg) never failed on bipartite graphs and failed

three times on general graphs, for $c \in \{3.3, 3.8, 8.2\}$ with failure rate $\lambda = 1/100$. For the other algorithms we observed the following behavior.

- For general graphs at $c = 2.8$ all of them have a failure rate λ of at least 0.86. For OPT(1,2):HEU(deg,deg) we could replicate the behavior, observed in [11], that for $c \leq 2.6$ and $c \geq 3.7$ the failure rate of the algorithm is almost zero while for the other values of c the failure rate is very high, reaching its peak with $\lambda = 1$ at $c = 3.0$. For the other heuristics λ stays quite high after $c = 2.8$.
- For bipartite graphs the situation is different. The failure rates go up only beyond 2.6 and the qualitative behavior varies widely among the different heuristics. For OPT(1,2):HEU(deg,deg) we observed a critical region of $2.8 < c < 3.5$ but with a less pronounced failure rate, reaching its peak at $c = 2.9$ with $\lambda = 0.37$. For all other algorithms the failure rate seems to increase for c beyond 8.

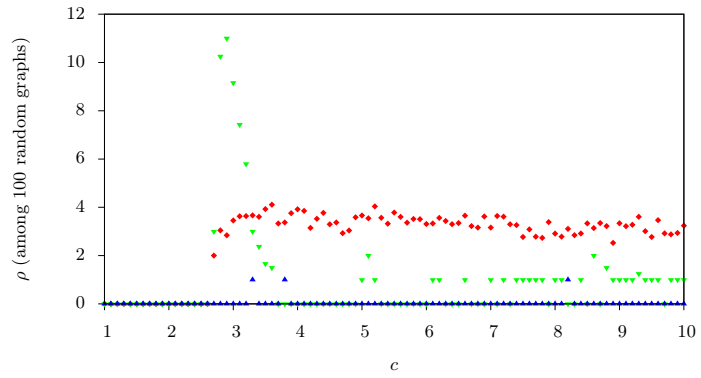
It is proven that OPT(1):HEU(rand) is likely to find a maximum matching for $c < e \approx 2.718$ mainly due to the optimal degree 1 reduction steps (so called e-phenomenon), see [1]. Our results indicate that including degree 2 reductions does not influence this bound much. Overall, the heuristics with degree 2 reduction more often give a maximum matching than their counterparts that can only utilize degree 1 reduction. In terms of the difference of the failure rates this effect is smallest for OPT(1):HEU(rand) and OPT(1,2):HEU(rand) on general random graphs. The best algorithms in terms of quality of solution are OPT(1,2):HEU(deg,deg) and OPT(1,2):HEU(pot,deg).

Edges Lost if Failure Occurs. Unlike before, we are only interested in the algorithms using degree 2 reduction, since on average they obtain the largest matchings. Figure 2 gives the average number of lost edges conditioned on the event that a failure occurs, for general and bipartite random graphs with $n = 10^6$ nodes and expected degree c ranging from 1 to 10. The legend for both plots is given on the top right of this paragraph. The figures for the number of lost edges for graphs with 10^4 and 10^5 nodes are given in Appendix B. The results are qualitatively similar to the results for $n = 10^6$.

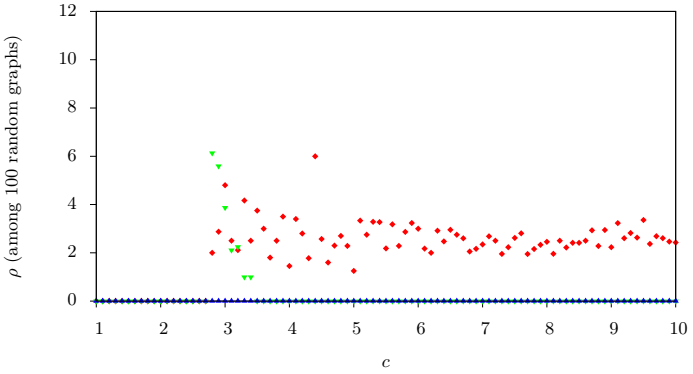


The mean over the values ρ for heuristic OPT(1,2):HEU(rand) is higher for the general graph scenario than for the bipartite graph scenario, while the variance of ρ is lower. The number of lost edges for heuristic OPT(1,2):HEU(rand) and for heuristic OPT(1,2):HEU(deg,deg), within their critical ranges, increases with increasing n, cf. Appendix B. Outside its critical range the double minimum degree heuristic OPT(1,2):HEU(deg,deg) loses mostly one edge on average for fixed c on general graphs and no edge on bipartite graphs. Our new algorithm OPT(1,2):HEU(pot,deg) loses one edge only in three cases.

Run-time Behavior. Figure 3 shows the average running time $\bar{t}$ of algorithm OPT(1,2):HEU(pot,deg) for calculating a matching, as well as the corresponding average fraction of degree 1 reduction steps $\overline{\#o1}$, degree 2 reduction steps



(a) general graphs

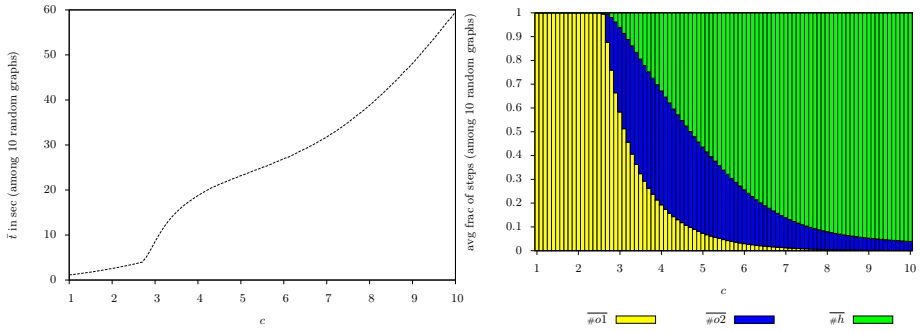


(b) bipartite graphs

Fig. 2. Average number of lost edges (if $\lambda > 0$) for graphs with $n = 10^6$ nodes.

$\overline{\#o2}$, and heuristic steps $\overline{\#h}$, on general random graphs with 10^6 nodes. The run-time behavior on bipartite random graphs of this size is qualitatively and quantitatively quite similar and given in Appendix C. The failure rate was zero in these experiments.

The average running time exhibits a non-linear increase. In a first phase, for $1 \leq c \leq 2.8$, the slope is linear and quite low. This is because in this range the running time is dominated by the fraction of degree 1 reduction steps $\overline{\#o1}$ which is more than 99 percent. It follows a second phase starting with a sudden increase of $\bar{t}$ which starts to flatten soon at c about 3.5. This goes along with a strong decrease of $\overline{\#o1}$ and increase of $\overline{\#o2}$ and $\overline{\#h}$. The next slight increase of the slope seems to be between $c = 6$ and $c = 7$ when $\overline{\#o1}$ falls below 0.03 and the fraction of heuristic steps $\overline{\#h}$ is more than 0.7, which indicates the begin of a third phase. The slope in this phase is larger than in the first phase and seems to be slightly non-linear. The sample variance of the running time is very low for the first phase and then increases slightly with increasing c; we observed a maximum of about 0.29 for general random graphs and of about 0.38 for bipartite random graphs.



(a) average running time

(b) average fraction of steps (stacked histogram)

Fig. 3. Run-time behavior of algorithm OPT(1,2):HEU(pot,deg) on general random graphs with 10^6 nodes.

4 Summary and Future Work

We proposed a new greedy algorithm to solve the maximum cardinality matching problem on random graphs with constant expected degree c, and found in experiments that this algorithm has a very low failure rate for a broad range of c. It is an open problem to prove that this behavior is to be expected.

The algorithm itself is an adaption of the selfless algorithm of Sanders [14] for orienting graphs, which was successfully generalized to orienting hypergraphs before, see [6]. It seems possible that the "selfless approach" can be used as generic building block for other greedy algorithms on random graphs too, like, e.g., graph coloring, which would be interesting to investigate.

References

1. Aronson, J., Frieze, A.M., Pittel, B.: Maximum matchings in sparse random graphs: Karp-Sipser revisited. Random Struct. Algorithms 12(2), 111–177 (1998)
2. Bast, H., Mehlhorn, K., Schäfer, G., Tamaki, H.: Matching Algorithms Are Fast in Sparse Random Graphs. Theory Comput. Syst. 39(1), 3–14 (2006)
3. Cain, J.A., Sanders, P., Wormald, N.C.: The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In: Proc. 18th SODA. pp. 469–476. SIAM (2007)
4. Chebolu, P., Frieze, A.M., Melsted, P.: Finding a maximum matching in a sparse random graph in $O(n)$ expected time. J. ACM 57(4) (2010)
5. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. J. Symb. Comput. 9(3), 251–280 (1990)
6. Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., Rink, M.: Tight Thresholds for Cuckoo Hashing via XORSAT. In: Proc. 37th ICALP (1). pp. 213–225. LNCS, Springer (2010)

7. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics 17, 449–467 (1965)
8. Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., Rossi, F.: GNU Scientific Library Reference Manual - Edition 1.15, for GSL Version 1.15 (2011), http://www.gnu.org/software/gsl/manual/
9. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. SIAM J. Comput. 2(4), 225–231 (1973)
10. Karp, R.M., Sipser, M.: Maximum Matchings in Sparse Random Graphs. In: Proc. 22nd FOCS. pp. 364–375. IEEE Computer Society (1981)
11. Magun, J.: Greedy Matching Algorithms: An Experimental Study. ACM Journal of Experimental Algorithmics 3, 6 (1998)
12. Micali, S., Vazirani, V.V.: An $O(\sqrt{|v|}\cdot|E|)$ Algorithm for Finding Maximum Matching in General Graphs. In: Proc. 21st FOCS. pp. 17–27. IEEE Computer Society (1980)
13. Mucha, M., Sankowski, P.: Maximum Matchings via Gaussian Elimination. In: Proc. 45th FOCS. pp. 248–255. IEEE Computer Society (2004)
14. Sanders, P.: Algorithms for Scalable Storage Servers. In: Proc. 30th SOFSEM. pp. 82–101. LNCS, Springer (2004)
15. Thomas, D.B., Luk, W., Leong, P.H., Villasenor, J.D.: Gaussian random number generators. ACM Comput. Surv. 39 (2007)
16. Vazirani, V.V.: A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm. Combinatorica 14(1), 71–109 (1994)

A Failure Rates

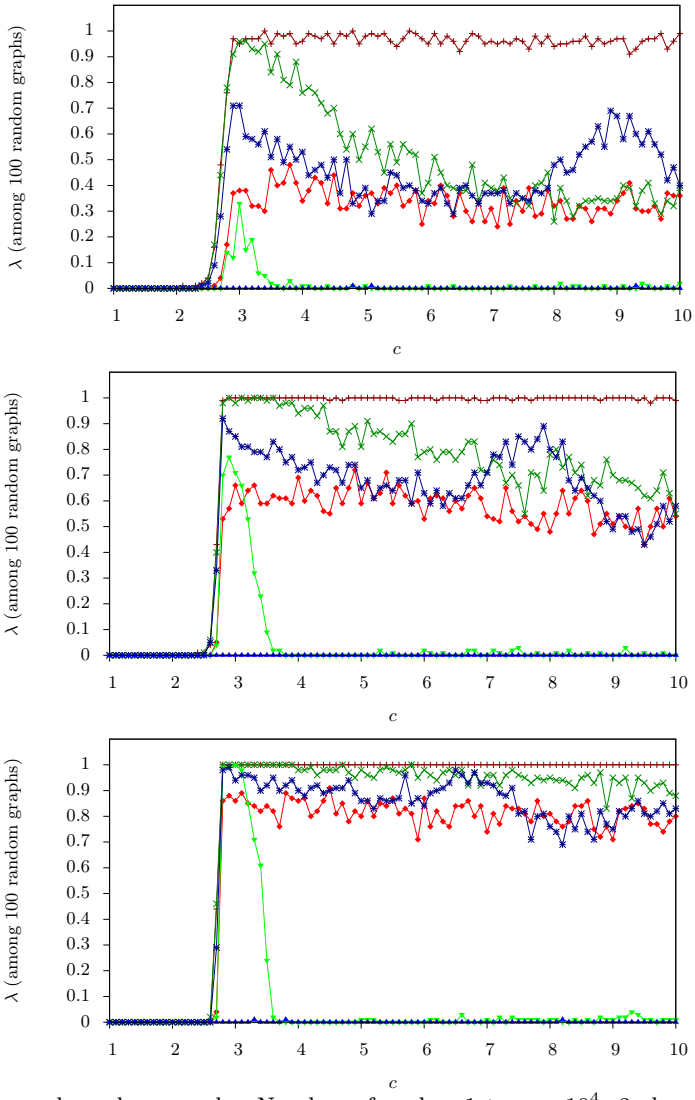


Fig. 4. General random graphs. Number of nodes: 1st $n = 10^4$, 2nd $n = 10^5$, 3rd $n = 10^6$.

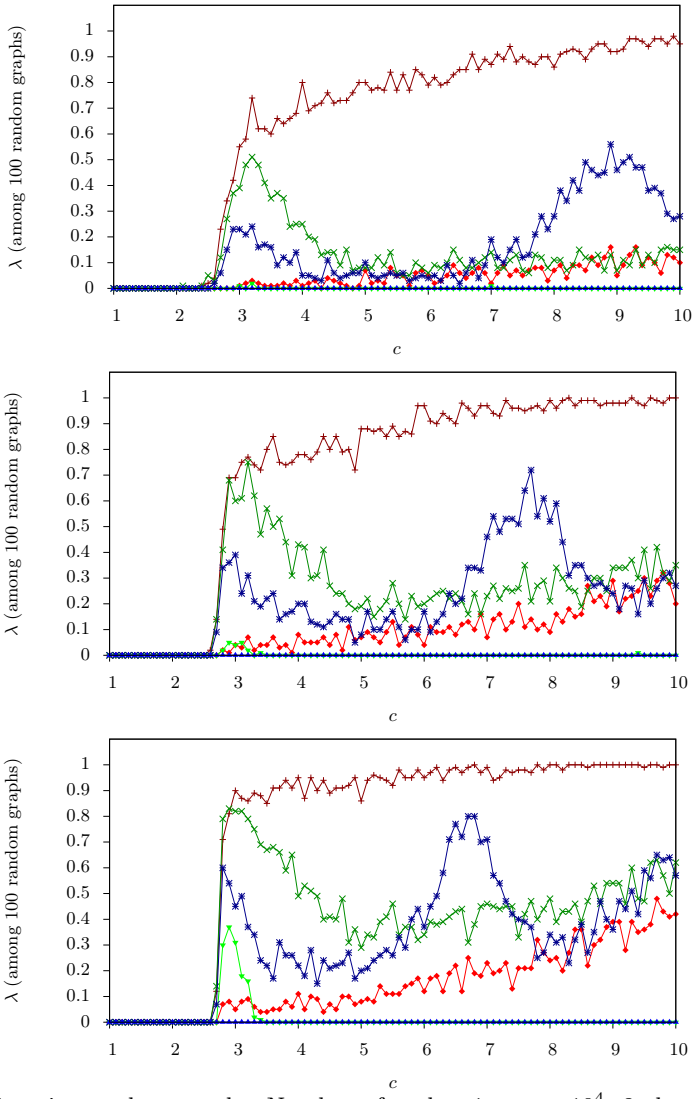


Fig. 5. Bipartite random graphs. Number of nodes: 1st $n = 10^4$, 2nd $n = 10^5$, 3rd $n = 10^6$.

B Average Number of Lost Edges if Failure Occurs

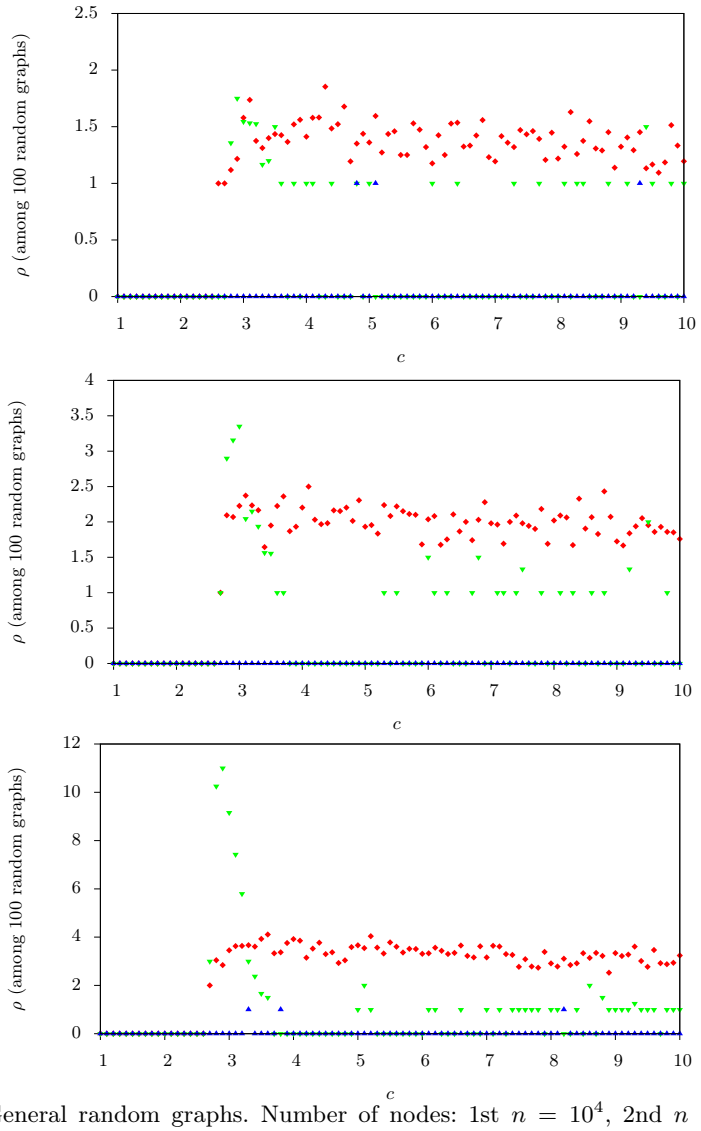


Fig. 6. General random graphs. Number of nodes: 1st $n = 10^4$, 2nd $n = 10^5$, 3rd $n = 10^6$.

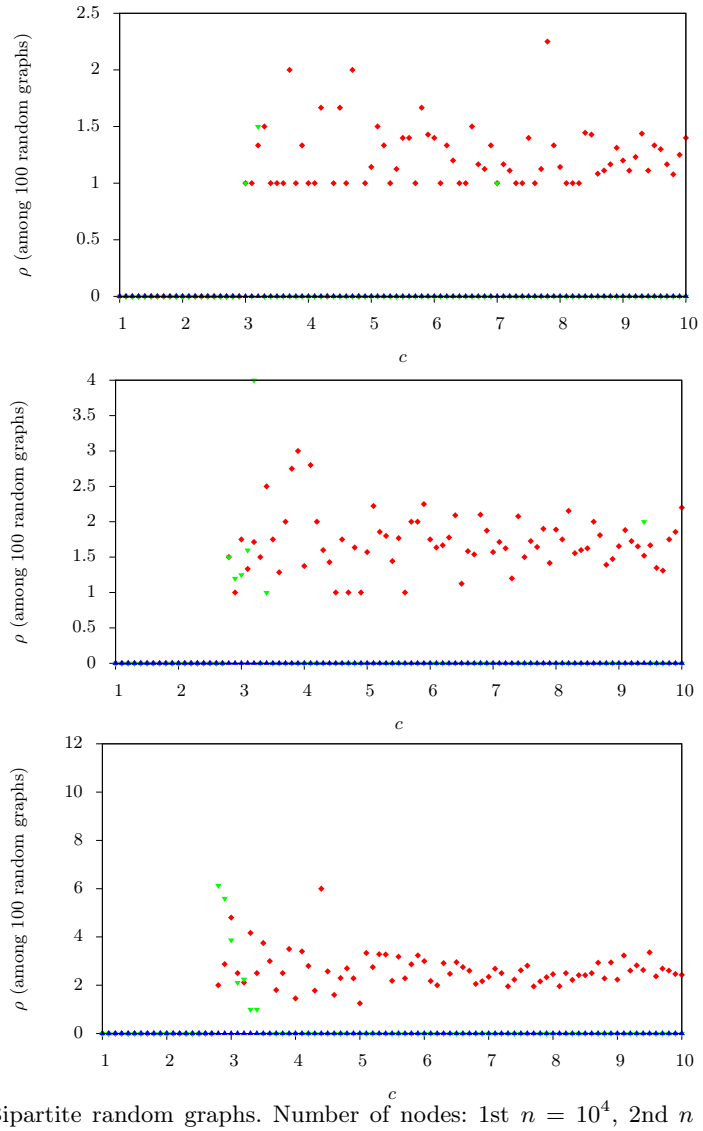
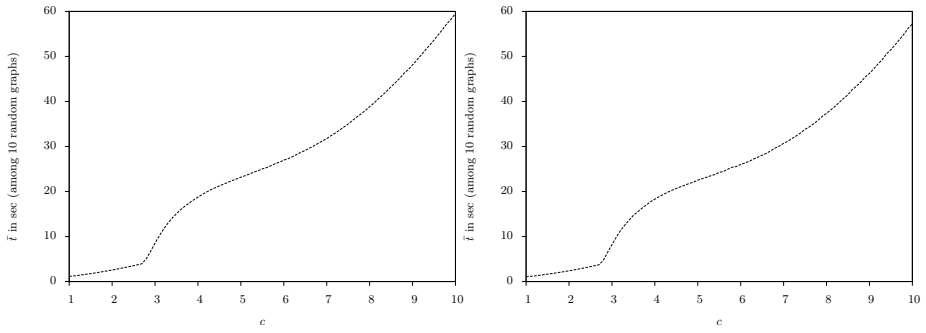


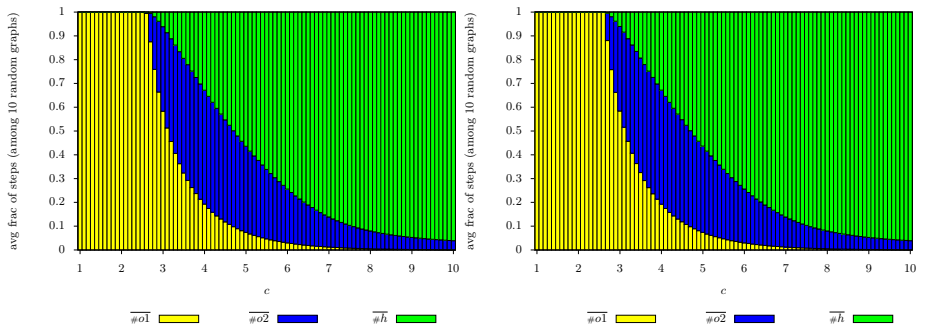
Fig. 7. Bipartite random graphs. Number of nodes: 1st $n = 10^4$, 2nd $n = 10^5$, 3rd $n = 10^6$.

C Average Running Times and Average Fraction of Steps



(a) general graphs (b) bipartite graphs

Fig. 8. Average running times in seconds for OPT(1,2):HEU(pot,deg) to obtain a matching on random graphs with $n = 10^6$ nodes.



(a) general graphs (b) bipartite graphs

Fig. 9. Stacked histogram of average fraction of degree 1 reduction steps, degree 2 reduction steps, and heuristic steps, of algorithm OPT(1,2):HEU(pot,deg) on random graphs with $n = 10^6$ nodes.